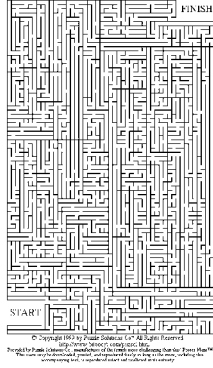


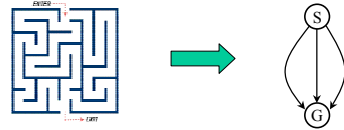
Searching

- Assume goal- or utility-based agents:
 - state information
 - ability to perform actions
 - goals to achieve
- Next task to achieve is to determine the "best" path to the goal



Search Domains

- Searching is the act of navigating through various agent states, until it reaches the goal state
 - set of possible intermediate states is called the **state space**
- In general, searching problems assume a finite set of states
 - limit search domain to realistic boundaries
 - reduce the granularity of state measurements

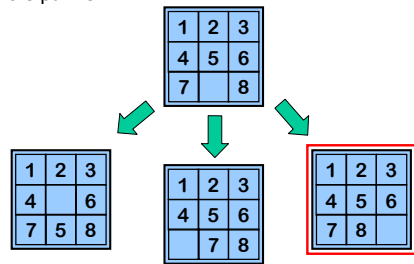


Search Domains (cont'd)

- State space is determined by the possible values that the agent's state can have, as a result of the actions that can be performed
 - Must have an **initial state** and one or more **goal states**
- Usually drawn as a graph, with nodes representing the possible states and edges representing the transitions between them
 - Each transition has a **path cost** associated with it. Path costs are generally non-negative, and assumed to be 1 if not specified.
 - A **successor function** produces all the sets of available actions and subsequent states, given a current state.
- A solution consists of the path from initial state to goal state.

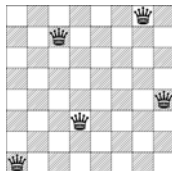
Searching Examples

- 8-tile puzzle



Searching Examples

- 8-Queens puzzle
 - place 8 queens on a chessboard so that no two queens attack each other.
 - purely generate-and-test: $\sim 64^8$ solutions
 - using successor functions: $\sim 8^8$ possible states to explore
 - n-queens** problem: place n queens on an $n \times n$ board



Searching Examples

- SAT Problem
 - Made of **literals** and **clauses** in **conjunctive normal form**
 - literal = propositional symbol or its negation (e.g. P , $\neg P$)
 - clause = disjunction of literals; 3-clause is a disjunction of 3 literals (e.g. $P \vee Q \vee \neg R$)
 - conjunctive normal form = conjunction of clauses.
 - Example: $(P \vee Q \vee \neg R) \wedge (\neg P \vee Q \vee R) \wedge (\neg P \vee \neg R \vee \neg S) \wedge (P \vee \neg S \vee T)$
 - All **satisfiability** problems can be represented as SAT problems
 - Searching all possible solutions is NP complete.

Search Criteria

- When evaluating a search technique, use the following metrics:
 - Completeness:** Does the search algorithm guarantee a solution, if one exists?
 - Optimality:** Does it find the best solution, if multiple solutions exist?
 - Efficiency:** What kind of time and/or space does it take in finding a solution?
- Treat search space like maze or road map; assume the ability to **backtrack** to previous forks in search space, in dead end cases
- Searching produces **search tree** (no loops)

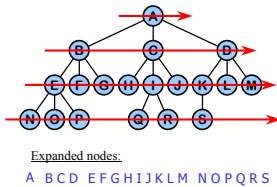
Uninformed Searches

- A search strategy is called **uninformed** if it has no other knowledge about the search space, other than the successive states at each stage
 - i.e. plotting a course without a map



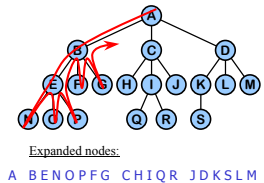
Breadth-first Search

- Breadth-first search** expands the start state first, and then expands successive states, one level at a time.
- Slow, but guaranteed to find the closest solution (if one exists)
- Complexity analysis:
 - Time: $O(b^d)$
 - Space: $O(b^d)$
 - (b is the branching factor, the maximum number of successor states possible. d is the depth of the solution in the search tree)
- Like level-order traversal



Depth-first Search

- Depth-first search** expands states down a single branch of the search tree until the goal or a dead end is reached (requires backtracking)
- Faster, but dangerous. Can explore far past solution depth, and the first solution isn't guaranteed to be the best possible.
- Complexity analysis:
 - Time: $O(b^L)$
 - Space: $O(bL)$
 - (L is the maximum reasonable search depth)
- Like preorder traversal

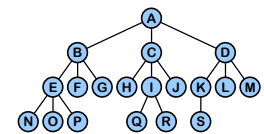


Uniform Cost Search

- Breadth-first search assumes that all transitions have a uniform path cost.
- Uniform cost search** expands unexplored states with the lowest path cost from the start state
- Given uniform path costs, this variation behaves just like breadth-first.
- Originated from Dijkstra's algorithm for finding the past cost between two points

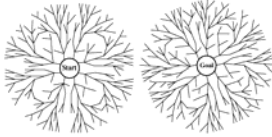
Iterative Deepening

- Depth-first search can overshoot the solution depth or produce a sub-optimal solution
- Iterative deepening attempts to combine the solution-finding benefits of breadth-first with the space efficiency of depth-first.
 - Perform depth-first search to a limited depth L (e.g. 3)
 - If solution is not found, repeat with larger L (e.g. 6)
- Complexity analysis:
 - Time: $O(b^d)$
 - Space: $O(bd)$
- Wastes some time and space, but not too much



Backwards & Bidirectional Search

- Variations of breadth or iterative deepening search
 - Backward search** = begin search at goal state, end at start state. Doesn't work as well with multiple goal states, and branching factor might be different in reverse.
 - Bidirectional search** = Begin search at start and goal states simultaneously, search until common state is found.
 - Time & space complexity: $O(b^{d/2})$



Searching with Uncertainty

- Two possible problems can arise when searching:
 - incomplete knowledge of states (**sensorless problem**)
 - uncertain outcome of actions (**contingency problem**)
- Sensorless problems produce a belief state space, where each belief state is the set of possible states that the agent could be in (see Figure 3.21 in text)
 - must find set of actions that lead to a belief state that contains nothing but goal states in it
- Contingency problems arise when the outcome of an action can be one of several possible states
 - agent must behave sensibly for whatever new state it is in, even if this makes planning difficult

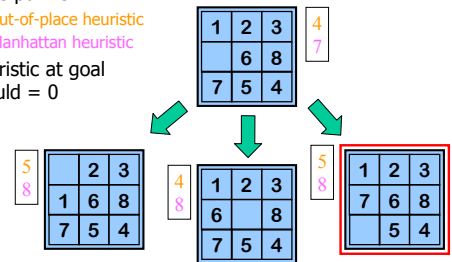
Heuristic Searches

- Also known as **informed searches**
- A **heuristic** is a "rule of thumb" that allows an agent to estimate the distance between a particular state and the goal
- Heuristic function** = estimated cost of path from current state n to goal state $\rightarrow h(n)$
 - e.g. **straight-line heuristic**: direct distance between current position and destination on a map
- Heuristic searches ignore path cost information, and focus instead on seeking the solution



Heuristic Example

- 8-tile puzzle
 - out-of-place heuristic**
 - Manhattan heuristic**
- Heuristic at goal should = 0



Greedy Search

- Greedy searches** look at the "best" available states, given an appropriate heuristic function
- Best-first search** explores the state that is closest to the goal, according to the heuristic
 - Also known as **gradient descent** or **hill-climbing** algorithms
 - Advantage**: space usage is $O(1)$
 - Disadvantage**: can get stuck at a local minimum
- Variations to greedy search:
 - Beam search** = search the m most promising nodes
 - Gradient with backup** = allow the search to backtrack if stuck
 - Overall best-first** = choose best state to explore across entire unexplored frontier

A* ("A-star") Search

- Combines current path cost with estimated remaining past cost (e.g. MapQuest)
- A*** is a best-first search on $f(n) = g(n) + h(n)$
 - $f(n)$ is A* value for state n
 - $g(n)$ is the path cost to get to state n
 - $h(n)$ is the heuristic cost of cheapest estimated solution
- In order for a search using A* to return the optimal result, $h(n)$ must be an **admissible** heuristic, meaning that it must never overestimate the distance to the goal (always optimistic estimate)
 - optimistic estimate ensures that the states along the solution path always have a value smaller than the optimal solution

A* Search (cont'd)

- To use graph searching algorithms with A* to obtain optimal paths, $h(n)$ should also be **consistent**
 - consistency: $h(n) \leq g(n, n') + h(n')$ $\leftarrow n, n'$ are on solution path
 - consistent heuristics should not overestimate any segment of the path \rightarrow leads to wrong choices at that segment
 - consistent heuristics are always admissible (stricter than admissability, since consistent heuristics underestimate each segment in the solution path, whereas admissible heuristics apply to the entire path)

Avoiding Local Maxima

- Local maxima occur when a state space does not gravitate towards a single solution, but has smaller "pockets" that a heuristic search might be drawn to
- Requires techniques to "escape" from these pockets, to determine if they are the true best solution
 - **Simulated annealing** = allows a change in the variables, according to a specified **temperature** and variable **energy**
 - **Genetic algorithms** = randomly interchange variables of different solutions, or **mutate** certain variables
 - Typically only applicable to domains like SAT

